Relational Rippling: a General Approach

Relational Rippling: A General Approach *

Alan Bundy and Vincent Lombart
Department of Artificial Inl.diligence, University of Edinburgh,
80 South Bridge, Edinburgh, EH1 1HN, Scotland. Email: {bundy,vincent}aisb.ed.ac.uk

Abstract

We propose a new version of rippling, called *relational rippling*. Rippling is a heuristic for guiding proof search, especially in the step cases of inductive proofs. Relational rippling is designed for representations in which value passing is by shared existential variables, as opposed to function nesting. Thus relational rippling can be used to guide reasoning about logic programs or circuits represented as relations.
We give an informal motivation and introduction to relational rippling. More details, including formal definitions and termination proofs can be found in the longer version of this paper, [Bundy and Lombart, 1995].

Keywords: Rippling, heuristics, inductive proof, automated theorem proving, logic program transformation.

1 Introduction

Rippling is a heuristic technique for controlling search during automatic theorem proving, [Bundy *et al*., 1993]. It was originally developed for inductive theorem proving. Its role is to manipulate the induction conclusion to make it more like the induction hypothesis, thus enabling the hypothesis to prove the conclusion. Rippling can also be used for non-inductive proofs whenever a problem can be solved by reducing a syntactic difference between it and some previously solved problem.

"Part of the research reported in this paper was conducted while the first author was a visitor at the Max Planck Institut fur Informatik in Saarbriicken. He would like to thank his hosts, Harald Ganzinger and David Basin for inviting him to MPI and making his stay so pleasant. We would also like to thank David Basin, Toby Walsh, Helen Lowe, Julian Richardson and members of the mathematical reasoning group at Edinburgh for discussions about earlier versions of the ideas described here. This work is part of a project supported by SERC grant GR/H/23610, ESPRIT BRP grant 6810 and ARC grant 438. The second author is supported by the Belgian National Fund for Scientific Research, and his stay in Edinburgh is supported by HC&M Logic Program Synthesis and Transformation.

Rippling works by identifying the syntactic difference between the current problem and the previous one, [Basin and Walsh, 1993], and then moving that difference through nested functions to a place where it no longer prevents a match between them. Following Boyer and Moore, this matching process is called *fertilization.* Rippling is predicated on the assumption that value passing is done via function nesting. But there is a popular alternative technique for value-passing: via existentially quantified shared variables. This is the technique used, for instance, in logic programming. It is also used in the relational representation of circuits. There is thus a strong motivation to adapt the ideas of rippling to situations in which value passing is done via shared variables between relations, instead of nested functions. We will call this adapted rippling, *relational rippling.* When we need to draw a distinction, we will call the original rippling: *functional rippling*.

Our goal is to implement relational rippling in a proof planning environment, [Bundy, 1988]. This entails devising relational rippling tactics to guide a proof editor like *Oyster* and devising methods to specify these tactics in a meta-logic for use by a proof planner like $CI^{A}M$, [Bundy *et al*., 1990].

Notational Conventions

We use upper case for variables (free or bound), lower case for constants and greek letters for meta-variables. Where variables are universally quantified at top level we will usually omit the quantifier. The notation *p(X)* means that *p* contains at least one free occurrence of *X*. It may also contain other variables.

We use => for the rewrite arrow and —+ for implication. Reasoning is backwards from goal to hypothesis. Therefore, when an implication, *A* —► B, is used as a rewrite rule on positions of positive polarity it is applied right to left, *i.e. B* => *A*.

2 Background

To make this paper self-contained we give a brief introduction to functional rippling and to value passing via shared existential variables.

2.1 Functional Rippling

Suppose we are allowed to use the hypothesis: $f_3(f_2(f_1(x)))$ in the proof of the goal[1]: $f_3(f_2(f_1(s_1(x))))$. Clearly these two formulae differ only in the inclusion of the function $s_1(\ldots)$ in the goal. This function is preventing matching the hypothesis against the goal. In inductive proofs, for instance, s_1 might be a step function inserted in the induction conclusion by the induction rule.

To initialise rippling, the two formulae are difference matched [Basin and Walsh, 1993]. We represent the result by annotating the goal. Those parts of the goal which differ from the hypothesis are put in boxes with a directional arrow. These are called *wave-fronts*. Any sub-expressions within these wave-fronts that do match the hypothesis are underlined. These are called *wave-holes*. In our abstract example the annotated expression is:

$$(1) \qquad f_3(f_2(f_1(\boxed{s_1(\underline{x})}^{\uparrow}))) \qquad \text{(1)}$$

Those parts of the expression which are outside wave-fronts or are inside wave-holes are called the *skeleton*. The skeleton of (1) is $f_3(f_2(f_1(x)))$. Note that this is the same as the induction hypothesis. The expression we get by removing all the wave-front annotation is called the *erasure*.

Rippling now proceeds to move the wave-fronts outwards in a series of rewriting steps:

$$f_3(f_2(f_1(\boxed{s_1(\underline{x})}^{\uparrow}))) \Rightarrow f_3(f_2(\boxed{s_2(\underline{f_1(x)})}^{\uparrow})) \Rightarrow f_3(\boxed{s_3(\underline{f_2(f_1(x))})}^{\uparrow}) \Rightarrow \boxed{s_4(\underline{f_3(f_2(f_1(x)))})}^{\uparrow}$$

Note that the skeleton is the same in each of these expressions. Only the wave-front moves. At the end of rippling a copy of the hypothesis is embedded in the goal. Fertilization is then able to rewrite the goal to: $s_4(true)$.

The rewrite rules that permit this rewriting are called *longitudinal wave-rules*. They are also annotated. In our abstract example these wave-rules have the form:

$$(2) \qquad f_i(\boxed{s_i(\underline{X})}^{\uparrow}) \Rightarrow \boxed{s_{i+1}(\underline{f_i(X)})}^{\uparrow} \qquad \text{(2)}$$

for $i = 1, 2, 3$. Note how each wave-rule moves the wave-front outwards one step, while preserving the skeleton. A strictly decreasing measure can be associated with the wave-fronts. This gives direction to the wave-rules and ensures termination of functional rippling. This measure is based on the depth of the wave-front in the skeleton — see [Basin and Walsh, 1994][§5] for details.

In rippling, not only must the left hand side of the rewrite rule match the erasure of a sub-expression of the current goal, the annotations in the rule must also match some in the goal. This additional condition forces the rewriting to happen in a desirable direction. It also significantly reduces the search space — usually to a single branch.

[1] In line with our notational conventions, the goal may contain other variables too, suppressed here for the sake of readability.

For more discussion about functional rippling see [Bundy *et al.*, 1993] and for a formal definition see [Basin and Walsh, 1994].

2.2 Value Passing via Shared Variables

In a functional programming language the step case of a recursive definition of list reversal, *rev*, might be:

$$rev([H|T]) = app(rev(T), [H])$$

Note how the value of *rev* on the right hand side is passed to *app* by nesting one function inside another. In Prolog the corresponding definition would look quite different:

```
rev([H|T],RL) :- rev(T,RT), app(RT,[H],RL).
```

The functions *rev* and *app* have become relations with an extra argument. More importantly for our purposes the method of value passing has changed. Instead of rev passing its value by function nesting, it does so via the shared variable RT. This shared variable is implicitly existentially quantified. The logical form of this Prolog clause is:

$$rev([H|T], RL) \leftarrow \exists RT.\ rev(T, RT) \wedge app(RT, [H], RL)$$

Shared variables are also sometimes used to represent electrical circuits. Electrical components are represented by relations and the wires between them by such shared variables. This has the advantage over the functional representation of circuits that it can more simply represent components with more than one output wire.

Some of the conjectures and wave-rules we will consider will be translations of equations. For instance, the equation:

$$f(s_1(X)) = s_2(f(X))$$

translates into the equivalence:

$$(3)\ \exists X'.\ s_1(X, X') \wedge p(X', Y') \leftrightarrow \exists Y.\ p(X, Y) \wedge s_2(Y, Y') \qquad \text{(3)}$$

where predicates p, s_1 and s_2 correspond to functions f, s_1 and s_2.

3 The Rippling-Past Method

The key phase in relational rippling is *rippling-past*. In functional rippling wave-fronts are passed up through nested functions. In rippling-past they move sideways through a conjunction.

Just as in the functional case we must impose an annotation on expressions. These annotations are used to: locate the right sub-expression to rewrite; locate the appropriate rule to rewrite it; restrict the rewriting to ensure it is moving in the right direction; and to ensure termination of rewriting.

3.1 Special Problems

In relational rippling we face special problems that are not present in functional rippling.

- Conjunction is both associative and commutative and existential quantification is both commutative and alpha convertable. It may be necessary to apply these rules in order to match a wave-rule to a goal. Moreover, the skeleton may be altered by these rules during rewriting. It is, therefore, necessary to define skeleton preservation modulo these rules and to take them into account during wave-rule application.

- Value passing by shared existential variables displaces the arguments of predicates in skeletons. Compare the arguments of the two *p* predicates in (3). These arguments need to be replaced when defining skeletons, in order to ensure preservation.
- There is a loss of directionality when functional expressions are translated into relational ones. For instance, the left hand side of (3) could equally well represent *S1(f(Y'))* as *f(s1(X))*. We need to annotate wave-rules in a way that imposes a sense of direction on them to prevent looping and ensure termination.
- Relational rippling needs to be supplemented with other processes. It needs to be initialised and it needs to be integrated with functional rippling.

For these reasons relational rippling has proved quite difficult to formalise. Before arriving at the proposals in this paper, we experimented with a number of alternatives, each of which was eventually rejected, often because of quite subtle problems.

3.2 Skeleton Preservation

Consider annotating the equivalence (3) as a wave-rule. We will want to hide the predicates S*1* and ,S2 as wave-fronts, but we also want to hide the existential quantification that connects them to the *ps*. This suggests:

$$\boxed{\exists X'.\ s_1(X,X') \wedge \underline{p(X',Y')}}^{\uparrow} \Rightarrow \boxed{\exists Y.\ \underline{p(X,Y)} \wedge s_2(Y,Y')}^{\uparrow}$$

However, note that the skeletons on each side are not preserved due to their different arguments. The problem is caused by the displacement of their arguments by the wave-fronts. These displaced arguments need to be replaced. To do this we annotate each argument position with the argument that has been displaced from it, *e.g.*

$$\boxed{\exists X'.\ s_1(X,X') \wedge p(X',Y')}^{\uparrow} \Rightarrow \boxed{\exists Y.\ p(X,Y) \wedge s_2(Y,Y')}^{\uparrow}$$

where the annotation appears in a box beneath the argument position. In the process of matching the wave-rule against the current goal, the meta-variables a and B are instantiated to the displaced arguments. In simple cases, they will be instantiated to *X* and *Y'*, respectively, but in more complex cases it might not be the case.

When forming the skeleton we not only remove wave-fronts but we replace each argument in the skeleton with its annotation. The skeletons on both sides of the wave-rule are both now $p(\alpha,\beta)$, *i.e.* the skeleton is preserved. For readability, these displaced argument annotations will be dropped when not needed.

3.3 Directionality

Note that (3) can also be annotated as a wave-rule in the reverse direction.

$$\boxed{\exists Y.\ \underline{p(X,Y)} \wedge s_2(Y,Y')}^{\uparrow} \Rightarrow \boxed{\exists X'.\ s_1(X,X') \wedge \underline{p(X',Y')}}^{\uparrow}$$

To prevent looping we require some additional annotation to give a direction to the rule. The intuition behind relational rippling is that the wave-fronts ripple past each part of the skeleton in turn. We see them as going into one argument and out of another. We can capture this intuition by annotating each argument position in the skeleton with an arrow. A downwards arrow means that the wave-front must go into this argument and upwards arrow means that the wave-front must go out of this argument. Arrows in wave-rule and goal must match. So (3) can be annotated in two ways:

(4)

$$\boxed{\exists X'.\ s_1(X,X') \wedge \underline{p(\overset{\downarrow}{X'},\overset{\downarrow}{Y'})}} \Rightarrow \boxed{\exists Y.\ \underline{p(\overset{\uparrow}{X},\overset{\uparrow}{Y})} \wedge s_2(Y,Y')}^{\uparrow}$$

(5)

$$\boxed{\exists Y.\ \underline{p(\overset{\uparrow}{X},\overset{\uparrow}{Y})} \wedge s_2(Y,Y')}^{\uparrow} \Rightarrow \boxed{\exists X'.\ s_1(X,X') \wedge \underline{p(\overset{\downarrow}{X'},\overset{\downarrow}{Y'})}}$$

If wave-rule (4) applies to a goal then the arrows in the goal will prevent the immediate application of wave-rule (5) to reverse the ripple.

Moreover, these annotations can be used to prove termination of relational rippling using ideas similar to [Basin and Walsh, 1994]. The skeleton of a goal defines a directed graph in which the nodes are the arguments of the predicates and an arc goes from node m to node n iff the skeleton contains a predicate with arguments $\overset{\downarrow}{m}$ and $\overset{\uparrow}{n}$. Wave-fronts move through this graph in the direction of the arcs. As long as it is acyclic this movement must eventually stop.

We can define a measure on annotated expressions in the following way. The *depth* of each node is the length of the longest path from that node. A wave-front is at node n iff n is an annotation on one of its arguments. The weight of a node is the number of wave-fronts at that node. The measure is a vector whose i^{th} element is the sum of the weights of nodes of depth i. Measures are well-founded by the lexicographic order, $\succ$, on the reversed vectors. For instance, the measure of the left-hand side of (4) is $[0,1]$ and the measure of the right-hand side is $[1,0]$. Under the lexicographic order $rev([0,1]) = [1,0] \succ [0,1] = rev([1,0])$. It is a defining property of wave-rules that they must be measure-decreasing. This ensures that wave-rule application is terminating when it is applied to goals whose skeleton graphs are acyclic.

Those directionality annotations can be added dynamically, *i.e.* the application of a wave-rule can add arrows to the skeleton. In this case, adding an arrow adds an arc to the graph, but acyclicity must be checked dynamically too.

3.4 Abstract Example of Rippling-Past

To illustrate rippling-past we translate the abstract functional wave-rules (2) into relational form:

$$(7)\quad \boxed{\exists X_i'.\ s_i(X_i,X_i') \wedge \underline{p_i(X_i',X_{i+1}')}}^{\uparrow} \Rightarrow \boxed{\exists X_{i+1}.\ \underline{p_i(X_i,X_{i+1})} \wedge s_{i+1}(X_{i+1},X_{i+1}')}^{\uparrow}$$

where $i = 1,2,3$. Some further examples of wave-rules are given in figure 5. To illustrate rippling-past we give the relational analogue of the functional rippling from §2.1 in figure 1.

(6)
$$\exists X_2', X_3'.\ \boxed{\exists X_1'.\ s_1(x_1, X_1') \wedge \underline{p_1(X_1', X_2')}}^{\uparrow} \wedge p_2(X_2', X_3') \wedge p_3(X_3', x_4')$$
$$\exists X_2, X_3'.\ p_1(x_1, X_2) \wedge \boxed{\exists X_2'.\ s_2(X_2, X_2') \wedge \underline{p_2(X_2', X_3')}}^{\uparrow} \wedge p_3(X_3', x_4')$$
$$\exists X_2, X_3.\ p_1(x_1, X_2) \wedge p_2(X_2, X_3) \wedge \boxed{\exists X_3'.\ s_3(X_3, X_3') \wedge \underline{p_3(X_3', x_4')}}^{\uparrow}$$
$$\exists X_2, X_3.\ p_1(x_1, X_2) \wedge p_2(X_2, X_3) \wedge \boxed{\exists X_4.\ \underline{p_3(X_3, X_4)} \wedge s_4(X_4, x_4')}^{\uparrow}$$

As the wave-fronts pass each shared variable they rename it. Each 'step predicate', s_i, relates a value, X_i, to its 'successor', X_i'. The hypothesis contains the X_i values and, initially, the goal contains only their successors, X_i'. As the s_i wave-front ripples past the successor, X_i', it replaces it with its 'predecessor', X_i. At the end of rippling-past the 'successors' are all replaced by their 'predecessors', which are the values in the induction hypothesis. The goal is now much more like the hypothesis than it was at the beginning. Note that the skeleton of the goal is preserved throughout the ripple and the measure decreases at each step.

Figure 1: Rippling-Past: An Abstract Example

3.5 The Transport Problem

Note that we have omitted some steps from the abstract ripple-past in figure 1 above. Consider, for instance, (6). After the ripple-past it will be left in the state labelled (8) in figure 2. To prepare this goal for the next ripple-past we need to rearrange the conjunction and the existential quantifiers into the state labelled (9). We call this the *transport problem.*

We have experimented with various solutions to this problem. One solution is to use a matching algorithm which builds in the associativity and commutativity of conjunction and the commutativity and alpha convertability of existential quantification. For instance, we can treat both goal and rule LHS as a set of conjuncts and rearrange them as required for the match. This is a straightforward solution but has two disadvantages:

- if it is necessary to justify such a match in the underlying logic then it must be unpacked into the various rule applications; and
- it does not scale up to a situation where other connectives are interleaved with the conjunctions.

For these reasons we have also experimented with the use of *attraction,* [Bundy and Welham, 1981], and *normalisation.* The variable to be rippled-past is identified and annotated. Rewrite rules are then applied which bring occurrences of this variable closer together while preserving the skeleton. These rules are based on associativity and commutativity of conjunction and commutativity of existential quantification. Wave-rules are kept in a normal form, also based on these rules. The subexpression to be rewritten is put in this normal form after collection. The major disadvantage of this approach is that the normal form is not canonical, so backtracking is theoretically required — although seldom needed in practice.

4 The Initialisation Method

The ripple-past method assumes that the goal contains wave-fronts. Unfortunately, after induction the induction conclusion will not contain any wave-fronts. The relational form of induction usually creates a step case of the form:

$$s_1(x_1, \textcircled{x_1'}),\ \phi(x_1) \vdash \phi(\textcircled{x_1'})$$

where we have annotated the 'successor(s)' of the induction variable with a distinguished variable marker: a solid circle. We will call $\textcircled{x_1'}$ an *initial candidate.* As with functional rippling, we will assume that this initial annotation is built into the induction and inherited into the step case.

Since the induction conclusion contains no wave-fronts, an initialisation phase is required to prepare for rippling-past. This will use wave-rules of the form:

(10) $s_1(X_1, \textcircled{X_1'}) \rightarrow$
$$p_1(\textcircled{X_1'}, X_2') \Rightarrow \boxed{\exists X_2.\ \underline{p_1(X_1, X_2)} \wedge s_2(X_2, X_2')}^{\uparrow}$$

which we will call *initialising wave-rules.* Initialising wave-rules introduce wave-fronts into the induction conclusion.

The defining characteristic of initialising wave-rules is that they reduce the number of initial candidates, while preserving the skeleton. Eventually the number of initial candidates can be reduced no more and initialisation terminates. A formal definition of initialising wave-rules, a well-founded measure and a termination proof for initialisation are given in [Bundy and Lombart, 1995]. Examples of initialising wave-rules are given in figure 3.

5 Hybrid Rippling

It is sometimes necessary to combine relational rippling with functional rippling. For instance, suppose we try to prove:

$$\exists X_2.\ p_2(X_1, X_2) \wedge p_1(X_2, X_3) \leftrightarrow \exists Y_2.\ q_2(X_1, Y_2) \wedge q_1(Y_2, X_3)$$

After the completion of rippling-past, the induction conclusion might take the form:

(11) $\exists X_2.\ p_1(x_1, X_2) \wedge \boxed{\exists X_3.\ \underline{p_2(X_2, X_3)} \wedge s_3(X_3, x_3')}^{\uparrow}$
$$\leftrightarrow \exists Y_2.\ q_1(x_1, Y_2) \wedge \boxed{\exists Y_3.\ \underline{q_2(Y_2, Y_3)} \wedge t_3(Y_3, x_3')}^{\uparrow}$$

(8) $\exists X_2', X_3'.\ (\boxed{\exists X_2.\ \underline{p_1(x_1, X_2)} \wedge s_2(X_2, X_2')}^{\uparrow} \wedge p_2(X_2', X_3')) \wedge p_3(X_3', x_4')$

(9) $\exists X_2, X_3'.\ (p_1(x_1, X_2) \wedge \boxed{\exists X_2'.\ s_2(X_2, X_2') \wedge \underline{p_2(X_2', X_3')}}^{\uparrow}) \wedge p_3(X_3', x_4')$

The problem here is to bring all the sub-expressions in (8) containing the variable, X_2', close together and put them in the same order as the corresponding parts of wave-rule (7). Brackets show how the conjunction is grouped. Note that $\exists X_2'$ switches from skeleton to wave-front and $\exists X_2$ does the reverse.

Figure 2: The Transport Problem

$$cons(H, L, \underline{\textcircled{L'}}) \rightarrow len(\underline{\textcircled{L'}}, N') \Rightarrow \boxed{\exists N.\ \underline{len(L, N)} \wedge s(N, N')}^{\uparrow}$$

$$cons(H, L, \underline{\textcircled{L'}}) \rightarrow rev(\underline{\textcircled{L'}}, R') \Rightarrow \boxed{\exists R.\ \underline{rev(L, R)} \wedge [\exists H\, L.\ cons(H, nil, H\, L) \wedge app(R, H\, L, R')]}^{\uparrow}$$

$$app(X, Y, \underline{\textcircled{Z}}) \rightarrow len(\underline{\textcircled{Z}}, K) \Rightarrow \boxed{\exists I, J.\ \underline{len(X, I)} \wedge \underline{len(Y, J)} \wedge +(I, J, K)}^{\uparrow}$$

Initialising wave-rules are readily obtainable from the step cases of recursive definitions and from certain lemmas. Compare these rules with the relational wave-rules in figure 5.

Figure 3: Example Initialising Wave-Rules

$$\exists X.\ \boxed{\exists Y.\ \underline{A(X, Y)}}^{\uparrow} \Rightarrow \boxed{\exists Y, \exists X.\ \underline{A(X, Y)}}^{\uparrow}$$

$$\boxed{\exists X.\ \underline{A(X)}}^{\uparrow} \leftrightarrow \boxed{\exists Y.\ \underline{B(Y)}}^{\uparrow} \Rightarrow \boxed{\forall X.\ \underline{A(X)} \leftrightarrow B(X)}^{\uparrow}$$

$$\boxed{\underline{A} \wedge C}^{\uparrow} \leftrightarrow \boxed{\underline{B} \wedge D}^{\uparrow} \Rightarrow \boxed{\underline{A} \leftrightarrow \underline{B} \wedge C \leftrightarrow D}^{\uparrow}$$

These longitudinal wave-rules are only a selection of those required.

Figure 4: Example Longitudinal Wave-Rules

This is not yet ready for fertilization because wave-fronts are still embedded within the skeleton. However, the various connectives above these wave-fronts are now behaving like functions which pass their values by nesting. So to prepare for fertilization we need only some rippling-out.

Some of the longitudinal wave-rules required to ripple-out (11) are given in figure 4. With the aid of these rules we can rewrite (11) to:

(12) $\boxed{\forall X_3.\ (\exists X_2.\ p_1(x_1, X_2) \wedge p_2(X_2, X_3)) \leftrightarrow (\exists Y_2.\ q_1(x_1, Y_2) \wedge q_2(Y_2, X_3)) \wedge s_3(X_3, x_3') \leftrightarrow t_3(X_3, x_3')}^{\uparrow}$

Strong fertilization reduces this to:

$$\forall X_3.\ true \wedge s_3(X_3, x_3') \leftrightarrow t_3(X_3, x_3')$$

which is, hopefully, easier to prove than the original conjecture.

Hybrid rippling is also necessary when the initial wave-front is functional, but becomes relational after some rippling. An example of hybrid wave-rule illustrating this transformation is (14).

6 Results

The relational rippling method described in this paper has not yet been completely implemented, but an updating of the implementation of [Lombart and Deville, 1994] (see §7.1) to cope with the new requirements is in progress. So in order to test our method we have applied it by hand to a range of examples. Below we explore one such example in detail. Some other examples on which we have tested relational rippling are listed in figure 6. We have yet to find a relational inductive theorem for which relational rippling is inappropriate. However, as with functional rippling, it is sometimes necessary to supplement relational rippling with unblocking techniques.

Our worked example is a lemma used in a (simplified) proof that the transitive closure of a relation preserves the Church-Rosser (CR) property.

The CR property is defined as:

$$r(X, Y) \wedge r(X, Z) \rightarrow \exists V.\ r(Y, V) \wedge r(Z, V)$$

where $r(X, Y)$ means X rewrites to Y in one step. We define the transitive closure of r with explicit length of the branch as:

$$r^*(0, X, Y) \leftrightarrow X = Y$$

$$r^*(s(N), X, Y) \leftrightarrow \exists V.\ r(X, V) \wedge r^*(N, V, Y)$$

where $r^*(N, X, Y)$ means X rewrites to Y in N steps. Those definitions give us the wave-rules in figure 5.

The lemma we prove is then:

$$r^*(N, A, C) \wedge r(A, B) \rightarrow \exists D.\ r^*(N, B, D) \wedge r(C, D)$$

This is proved by induction on n, with the induction conclusion:

$$r^*(\underset{\boxed{N}}{\boxed{\underline{s(N)}}^{\uparrow}}, \underset{\boxed{A}}{A}, \underset{\boxed{C}}{C}) \wedge r(\underset{\boxed{A}}{A}, \underset{\boxed{B}}{B})$$

$$\rightarrow \exists D.\ r^*(\underset{\boxed{N}}{\boxed{\underline{s(N)}}^{\uparrow}}, \underset{\boxed{B}}{B}, \underset{\boxed{D}}{D}) \wedge r(\underset{\boxed{C}}{C}, \underset{\boxed{D}}{D})$$

(13) $r(X,Y) \wedge r(X,Z) \Rightarrow \exists V.\ r(Y,V) \wedge r(Z,V)$

(14) $r^*(s(N), X, Y) \Rightarrow \exists V.\ r(X,V) \wedge r^*(N,V,Y)$

Wave-rules (13) and (14) are drawn from the Church-Rosser property and the transitive closure definition, respectively. Note that (14) is a hybrid wave-rule.

Figure 5: Example Relational Wave-Rules

Applying the definition of transitive closure (14) gives:

$$\exists V.\ r(A,V) \wedge r^*(N,V,C) \wedge r(A,B) \rightarrow \exists D.\ \exists V.\ r(B,V) \wedge r^*(N,V,D) \wedge r(C,D)$$

Applying the CR property (13) gives:

$$\exists V.\ \exists V'.\ r^*(N,V,C) \wedge r(V,V') \wedge r(B,V') \rightarrow \exists D.\ \exists V.\ r(B,V) \wedge r^*(N,V,D) \wedge r(C,D)$$

and weak fertilization is now possible, which gives:

$$\exists V.\ \exists V'.\ \exists D.\ r^*(N,V',D) \wedge r(C,D) \wedge r(B,V') \rightarrow \exists D.\ \exists V.\ r(B,V) \wedge r^*(N,V,D) \wedge r(C,D)$$

which should be readily provable by symbolic evaluation.

7 Related Work

Alternative versions of relational rippling have been proposed by Lombart & Deville and by Ahs & Wiggins. Only the first proposal has been fully implemented.

7.1 Lombart & Deville

In [Lombart and Deville, 1994], Lombart and Deville propose a version of relational rippling developed by analogy with functional rippling. Initially, they limited themselves to relational reasoning which was a direct translation of functional reasoning -- what we will call the pseudo-functional case. This led them to make a number of simplifying assumptions. For instance, they assume a uniqueness property of the step predicate in order to initialise rippling-past, and they limit themselves to formulae which are existentially quantified conjunctions.

Several improvements have been subsequently made to their proposal, which make it more robust and more powerful. New annotations have been added to ensure skeleton preservation. The directionality annotations they had developed are now dynamically added. The rippling-past process on which they focus has be embedded between an initialization phase and a final rippling-out. A termination measure has been defined.

The annotations used in [Lombart and Deville, 1994] are slightly different from those proposed here: the wave-fronts do not include the existential quantifiers, and the directionalities are implicit (because they are statically fixed), but this is only syntactic sugar. On the other hand, they do not have skeleton preservation annotations.

7.2 Ahs & Wiggins

Ahs and Wiggins were the first to propose a version of relational rippling in [Ahs and Wiggins, 1994]. Although they also motivate their work by analogy to functional rippling, they do not make the simplifying assumptions of Lombart and Deville, so their proposal is more general. In [Ahs, 1995] Ahs gives details of a wave-rule parser and illustrates it on some examples. However, he gives no formal definitions or theoretical account. This means that his parser is inherently *ad hoc;* there is no basis to judge whether it is correctly implemented. His wave annotation puts wave-fronts and wave-holes around some distinguished variables. It is not clear how these annotations guide the rippling process. For instance, what we call the transport problem is not addressed, nor is the problem of skeleton preservation or termination. It is not clear how the wave annotations could help solve these problems.

8 Conclusion

In this paper we have proposed a new version of rippling which can reason about representations which used shared existential variables instead of function nesting to pass values. Relational rippling consists of two phases: initialisation and rippling-past and it must sometimes be combined with functional rippling. Initialisation is fairly straightforward, so most of our attention has been directed to rippling-past.

We have defined some meta-level annotation to be added to object-level formulae. This annotation is used to restrict the application of rewrite rules and hence reduce search and ensure termination. Wave-rules must be both skeleton preserving and measure decreasing. These

$$\exists XY.\ +(X,Y,XY) \wedge +(XY,Z,XYZ) \leftrightarrow \exists YZ.\ +(Y,Z,YZ) \wedge +(X,YZ,XYZ)$$
$$\exists Z.\ app(X,Y,Z) \wedge len(Z,K) \leftrightarrow \exists I,J.\ len(X,I) \wedge len(Y,J) \wedge +(I,J,K)$$
$$even(X) \wedge even(Y) \wedge +(X,Y,Z) \rightarrow even(Z)$$
$$anc_1(X,Y) \leftrightarrow anc_2(X,Y)$$
$$anc_1(Old,Mid) \wedge anc_1(Mid,Young) \rightarrow anc_1(Old,Young)$$
$$r^*(N,X,Y) \wedge r^*(M,X,Z) \rightarrow \exists V.\ r^*(M,Y,V) \wedge r^*(N,Z,V)$$

anc_1 and anc_2 are two alternative predicates for the ancestor relation. The ancestor proofs and the CR property preservation have no functional equivalent. Details of some of these proofs can be found in [Bundy and Lombart, 1995].

Figure 6: Example Theorems Provable using Relational Rippling

properties are inherited by the expressions being rewritten.

In an extended version of this paper, [Bundy and Lombart, 1995], we give formal definitions of the various concepts introduced informally here: well annotated term; skeleton; erasure; well-founded measures for attraction rules and relational wave-rules; the wave-rule types; preconditions of the various methods; *etc.* These definitions have been used to show that each phase of rippling terminates, as does the process as a whole. They can also be used to give a formal specification of a program to parse rewrite rules as wave-rules and attraction rules and automatically annotate them. Finally, they can be used to specify the methods and tactics which will be needed to implement relational rippling in a proof planning context. Such formal definitions are badly needed and their absence has hampered previous attempts to implement relational rippling. As a result of this formal analysis the proposals given here are quite improved from previous proposals.

The proposals here have been hand tested on a range of examples, both abstract and concrete, drawn from the step cases of inductive proofs of relational theorems. So far, they have been very successful in guiding these proofs. These tests have confirmed that relational rippling dramatically reduces the search for a proof most of the time there is no branching despite a highly explosive search space. This reduction in the search space seems not to exclude the required proofs. Even when relational rippling fails, an analysis of the failure can suggest how to patch the proof.

It remains to complete the implementation of the proposals made here and to test them more extensively.

References

[Ahs and Wiggins, 1994] T. Ahs and G. A. Wiggins. Relational rippling for logic program synthesis and transformation, 1994. Presented at the Fourth International Workshop on Logic Program Synthesis and Transformation; available as DAI research paper, forthcoming.

[Ahs, 1995] T. Ahs. Relational rippling (working title), 1995. Ph.L. Thesis, Computing Science Department, Uppsala University, Sweden.

[Basin and Walsh, 1993] D. Basin and T. Walsh. Difference unification. In *Proceedings of the 13th IJCAI.* International Joint Conference on Artificial Intelligence, 1993. Also available as Technical Report MPI-I-92-247, Max-Planck-Institute fur Informatik.

[Basin and Walsh, 1994] D.A. Basin and T. Walsh. Annotated rewriting in inductive theorem proving. Technical report, MPI, 1994. Submitted to JAR.

[Bundy and Lombart, 1995] A. Bundy and V. Lombart. Relational rippling: a general approach. Research Paper forthcoming, Dept. of Artificial Intelligence, Edinburgh, 1995. Shortened version submitted to IJCAI-95.

[Bundy and Welham, 198l] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence,* 16(2):189-212, 1981. Also available from Edinburgh as DAI Research Paper 121.

[Bundy *et* a/., 1990] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on Automated Deduction,* pages 647-648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[Bundy *et* a/., 1993] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence,* 62:185-253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Bundy, 1988] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction,* pages 111-120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[Lombart and Deville, 1994] V. Lombart and Y. Deville. Rippling on relational structures. Research report, November 1994. Available as research report RR94-16, Departement d'ingenierie informatique, Universite catholique de Louvain, Belgium.